ECE 385

Spring 2021

Experiment 2

A Logic Processor

Wenhao Tan & Jiacheng Huang
ABI
Yanye Li

Introduction

The purpose of this lab is first to create a logic processor. The processor should be able to perform 8 different functions, including AND, OR, XOR, 1, NAND, NOR, XNOR, 0. There are also 4 different routings for the result, including (A, B), (A, F), (F, B), (B, A). The logic processor we created on breadboard can operate on 4 bits, while the one we created on FPGA can operate on 8 bits.

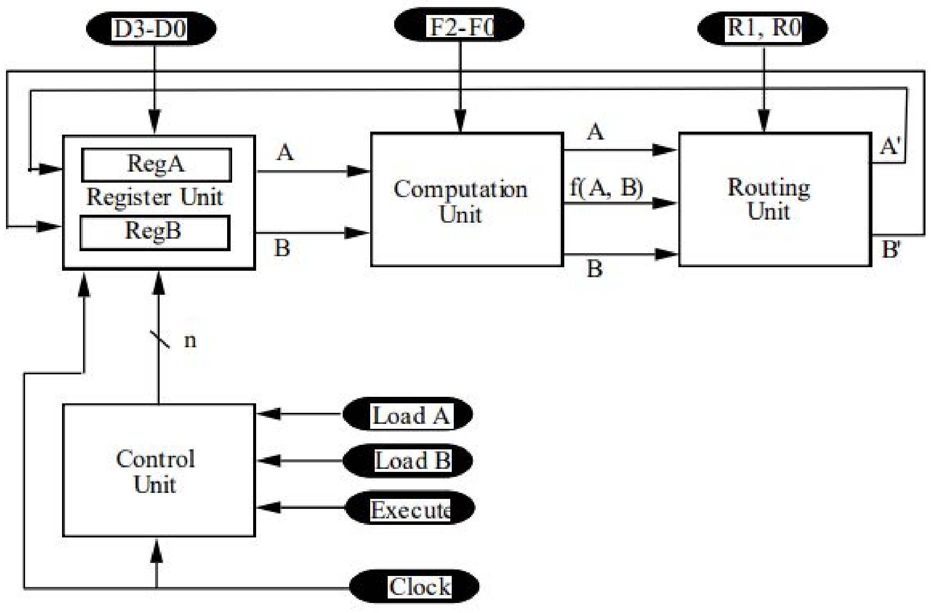
Operation of the Logic Processor

To load in data to the shift registers, we first have to flip the 4 switches that contain the data to the correct value. Then if we are loading in register A, we just have to flip the load A switch on for 1 clock cycle and it will parallel load all 4 bits into register A. Same thing for when we load in register B, we have to flip on load A switch for 1 clock cycle to parallel load all 4 bits into register B.
To initiate a computation and routing operation, we first have to flip the 3 switches controlling which function we want to operate and the 2 switches controlling which routing path we want our result to go to. Then we just have to switch on the Execute switch for at least 1 clock cycle, and the shifting process will complete in 4 clock cycles on its own.

Written Description of Circuit

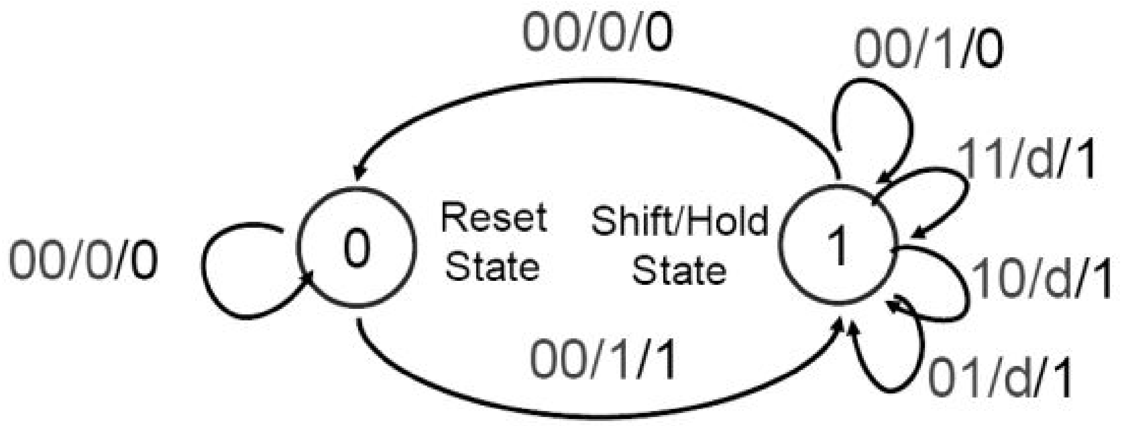
In this lab, we combined 2 shift register units, a computational unit, a routing unit, and a control unit to create a 4 bit logic processor. We used two CD74HC194E 4-bit shift registers with parallel loading. For the computational unit, we used a 4-to-1 MUX combining with an XOR gate and 3 switches to choose between the 8 different functions we want to generate. Similarly, for the routing unit, we used a 4-to-1 MUX and 2 switches to choose between the 4 different routing options. For the control unit, we used a SN74HC161N as our counter which counts from 00 to 11 and then reset because we need to shift right 4 times. We also used a flip-flop to store the current state we are in. Combining the Execution signal, the current state, and the 2 bit counter with some combinational logic, we are able to determine the next state, and if we should shift right.

Block Diagram



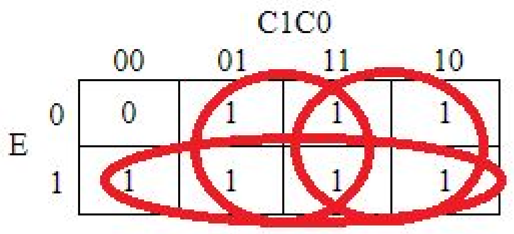
State Machine Diagram

We used a Mealy machine that consists only of two states, the reset state, and the shift/halt state. The reset state happens when the execute signal is low and the previous shift operation has been completed. Otherwise, the machine will stay in the shift/halt state.



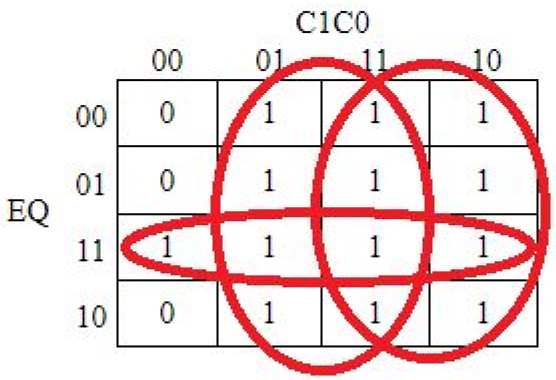
The first 2 bits in the diagram represents the value of our counter, the 3rd bit represents the state of the execution signal, and the 4th bit represents the shifting state.

Karnaugh Maps



E \ C1C0	00	01	11	10
0	0	1	1	1
1	1	1	1	1

Q+ = E + C1 + C0

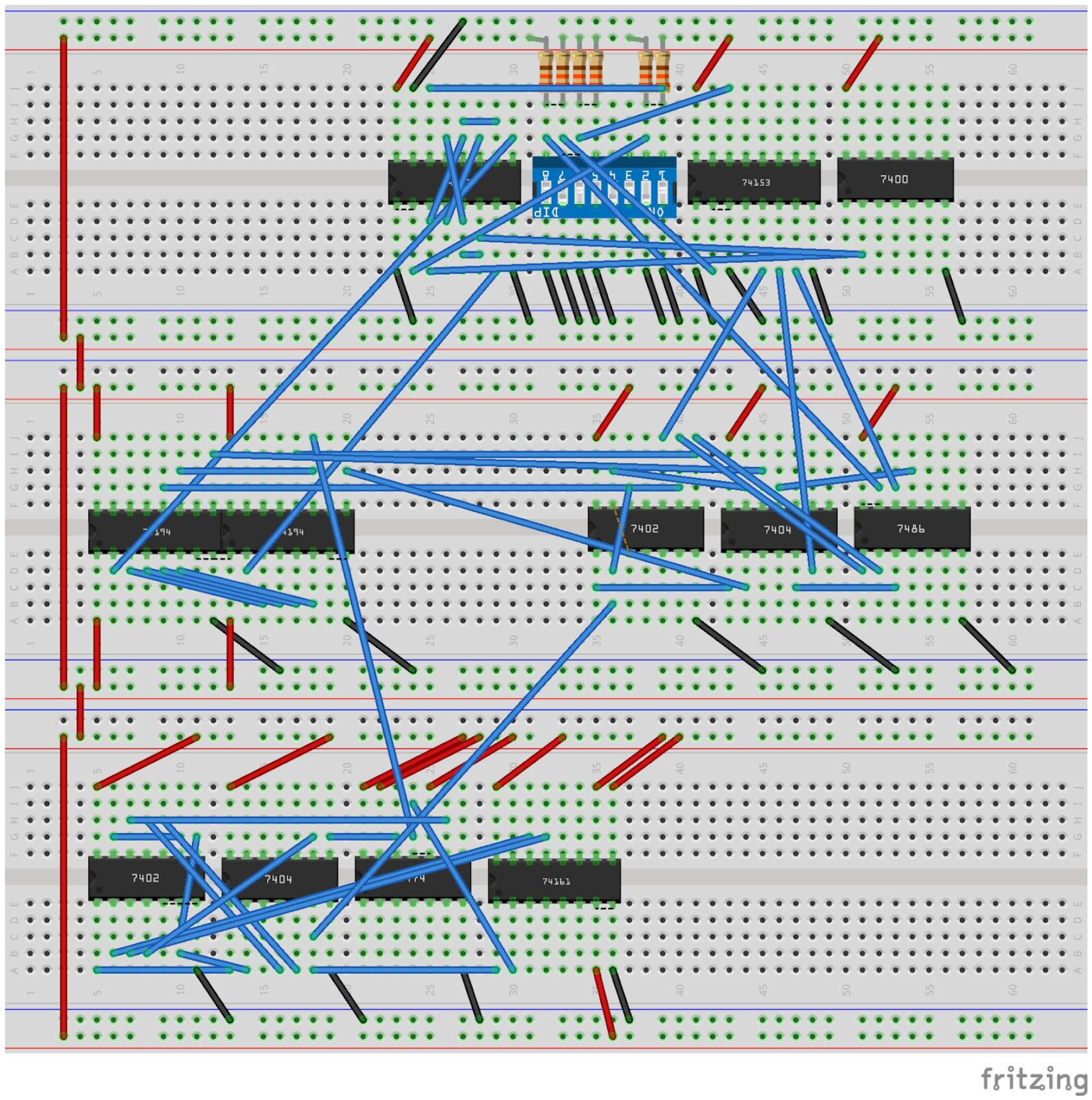


EQ \ C1C0	00	01	11	10
00	0	1	1	1
01	0	1	1	1
11	1	1	1	1
10	0	1	1	1

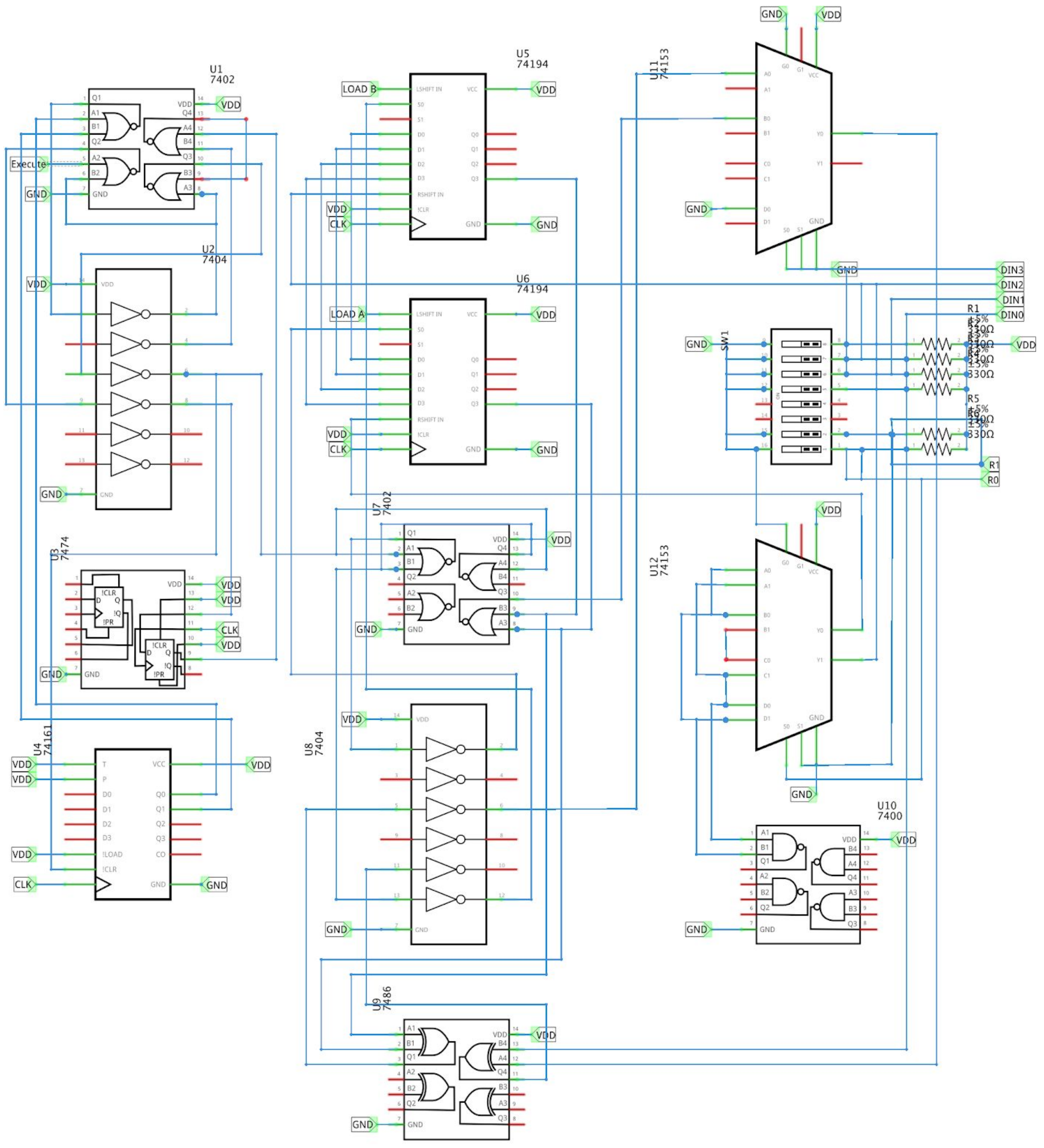
S = EQ + C1 + C0

After drawing out the K-maps for the next state Q+ and shift state S, we realized that our inputs only depend on the Execute signal E, the current state Q, and the 2 bit counter C1 and C0. Therefore, we built our control unit based on the boolean equations above using NOR and NOT gates chips because we do not have OR gates chips.

Breadboard View

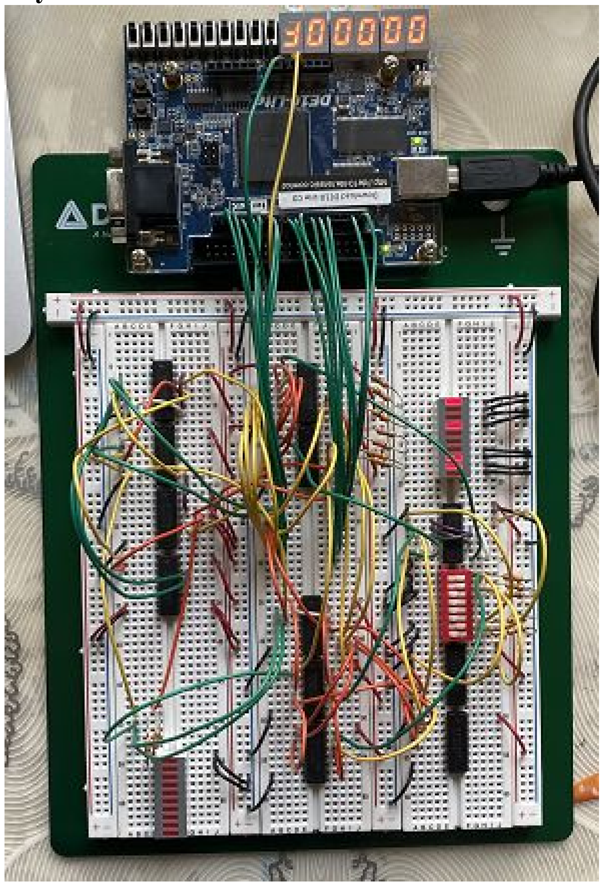


Circuit Schematic



fritzing

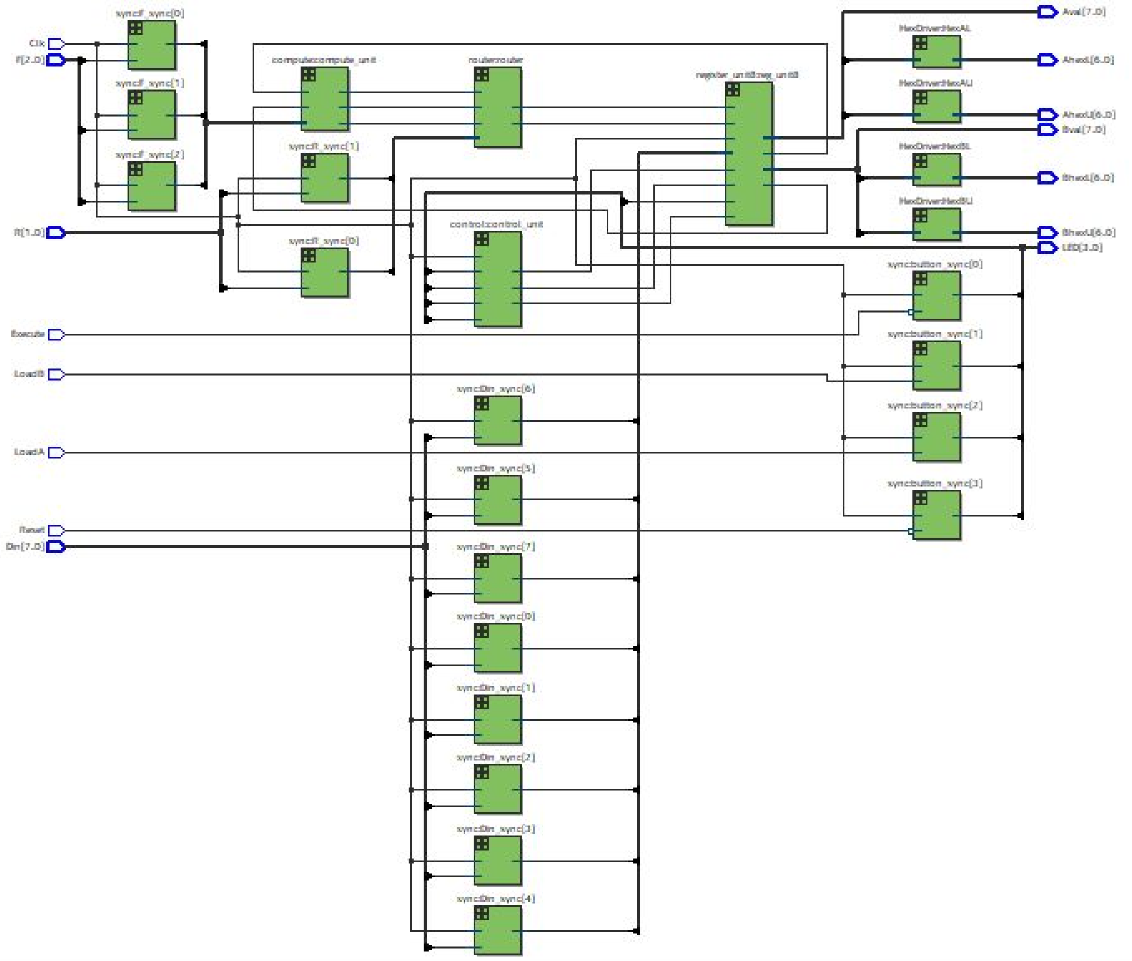
Physical BreadBoard Circuit



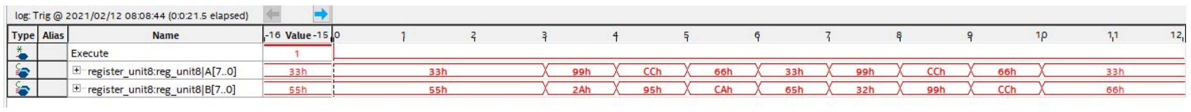
8-bit logic processor on FPGA

There are a few places we had to make changes to turn the 4-bit logic processor into an 8-bit logic processor on FPGA. We have to create a new module for an 8-bit shift register by combining 2 4-bit shift register modules together. Then we have to change the input and output from [3:0] to [7:0] for the data because we are working with 8-bits now. We also have to change the counter from 2-bits to 3-bits so it can count from 000 to 111. Lastly, we have to manually add 4 more states in the control module for the additional shift states.

RTL Block Diagram



SignalTap ILA Trace

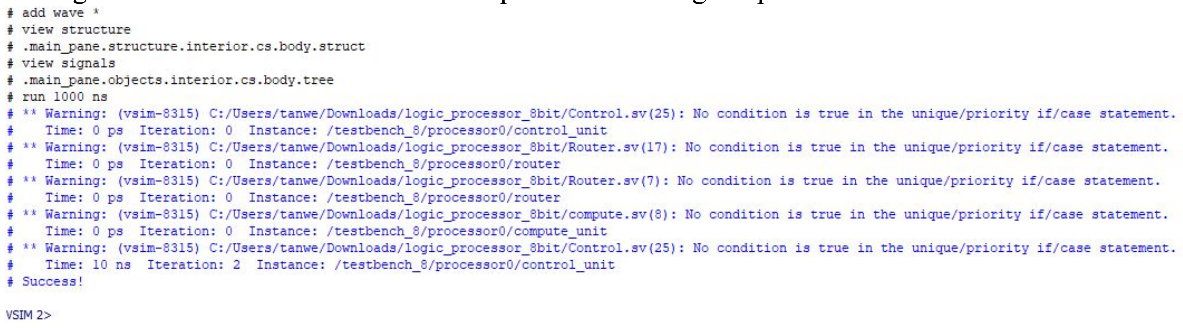


```
log: Trig @ 2021/02/12 08:08:44 (0:0:21.5 elapsed)
Type Alias Name                             -16 Value -15  0    1    2    3    4    5    6    7    8    9    10   11   12
           Execute                              1
           register_unit8:reg_unit8|A[7..0]     33h        33h            99h  CCh  66h  33h  99h  CCh  66h  33h
           register_unit8:reg_unit8|B[7..0]     55h        55h            2Ah  95h  CAh  65h  32h  99h  CCh  66h
```

We set up the SignalTap to trigger on Execute, meaning when we pressed down the Execute button, it would start collecting data at time 0. We chose to only capture the values inside our two shift registers. We can see from the figure that register A started with 8’h33, and register B started with 8’h55. We manually set the computational operation to be XOR and the routing to be (A, F), meaning register A will get back its original data while register B will contain the function result. As we can see, after exactly 8 clock cycles, the operation finishes and register A still contains 8’h33, while register contains 8’h66, which is correct because 8’h33 XOR 8’h55 = 8’h66.

RTL Simulation

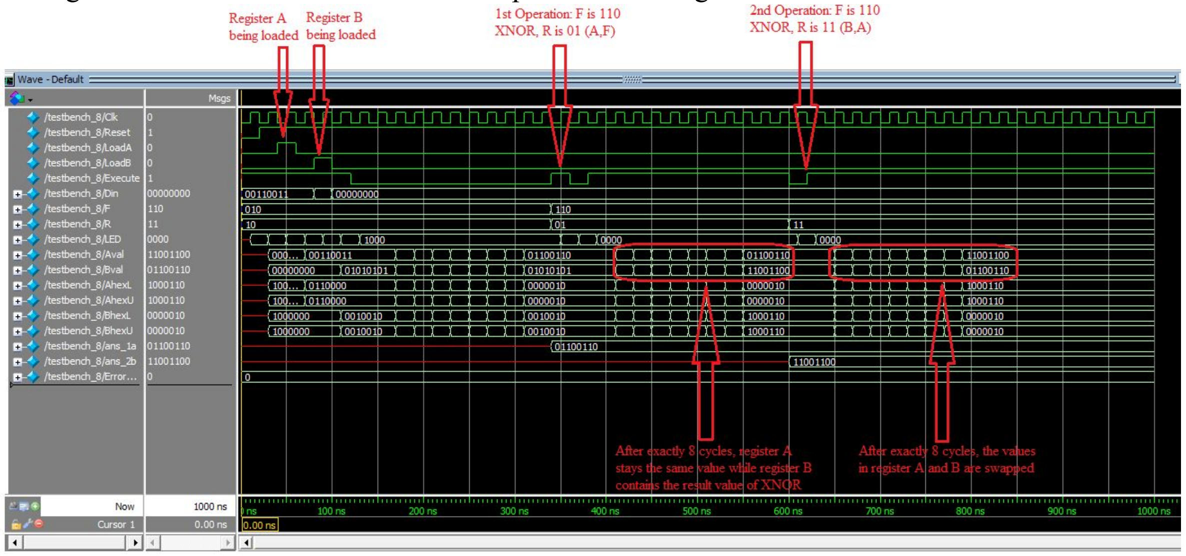
The figure below shows the successful output when running the provided testbench in ModelSim.



```
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run 1000 ns
# ** Warning: (vsim-8315) C:/Users/tanwe/Downloads/logic_processor_8bit/Control.sv(25): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench_8/processor0/control_unit
# ** Warning: (vsim-8315) C:/Users/tanwe/Downloads/logic_processor_8bit/Router.sv(17): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench_8/processor0/router
# ** Warning: (vsim-8315) C:/Users/tanwe/Downloads/logic_processor_8bit/Router.sv(7): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench_8/processor0/router
# ** Warning: (vsim-8315) C:/Users/tanwe/Downloads/logic_processor_8bit/compute.sv(8): No condition is true in the unique/priority if/case statement.
#    Time: 0 ps  Iteration: 0  Instance: /testbench_8/processor0/compute_unit
# ** Warning: (vsim-8315) C:/Users/tanwe/Downloads/logic_processor_8bit/Control.sv(25): No condition is true in the unique/priority if/case statement.
#    Time: 10 ns  Iteration: 2  Instance: /testbench_8/processor0/control_unit
# Success!

VSIM 2>
```

The figure below is the simulation wave output after running the testbench.


Register A being loaded
Register B being loaded
1st Operation: F is 110 XNOR, R is 01 (A,F)
2nd Operation: F is 110 XNOR, R is 11 (B,A)
Wave - Default
Msgs
After exactly 8 cycles, register A stays the same value while register B contains the result value of XNOR
After exactly 8 cycles, the values in register A and B are swapped
Now 1000 ns
Cursor 1 0.00 ns

	Msgs
/testbench_8/Clk	0
/testbench_8/Reset	1
/testbench_8/LoadA	0
/testbench_8/LoadB	0
/testbench_8/Execute	1
/testbench_8/Din	00000000
/testbench_8/F	110
/testbench_8/R	11
/testbench_8/LED	0000
/testbench_8/Aval	11001100
/testbench_8/Bval	01100110
/testbench_8/AhexL	1000110
/testbench_8/AhexU	1000110
/testbench_8/BhexL	0000010
/testbench_8/BhexU	0000010
/testbench_8/ans_1a	01100110
/testbench_8/ans_2b	11001100
/testbench_8/Error...	0

Bugs Encountered

The bugs we encountered while building the breadboard circuits in lab 2.1 are mostly simple, easy-fix errors like forgetting to connect ground or connecting to the wrong row. We were getting a bug that is outputting the incorrect computational function XNOR, but all other functions worked perfectly. We discovered that the problem was that we connected to the wrong side of the switch.

We also encountered some bugs in lab 2.2. The first bug was a compiler error saying we have undefined top-level design entity. This was easily fixed by assigning the processor.sv as the top-level design entity. Then, we were getting errors when running the RTL simulation and we found out that we did not add in the additional 4 shift states. Lastly, we had some bugs in SignalTap. First one being we did not set up a clock signal. Another error we made was we selected the wrong output signals to display. The names A, B are the same in the selection list, but the correct ones were in the register sublists.

Answers to Post-Lab Questions

Describe the simplest (two-input one-output) circuit that can optionally invert a signal (i.e., one input determines if the output is equal to the other input or equal to the other input inverted). Explain why this is useful for the construction of this lab.

- This is a simple 2-to-1 MUX. It is useful in the construction of lab 2.1 because we need to make a computational unit that contains 8 different operations. However, 4 of the operations are just the invert of the other 4. So with the help of this 2-to-1 MUX, instead of having to make a 8-to-1 MUX, we can just use a 4-to-1 MUX for half of the operations and invert the result for the other half.

Explain how a modular design such as that presented above improves testability and cuts down development time.

- The modular design above can sometimes save the amount of chips that we need to use in a design. For example, if we used an 8-to-1 MUX, we still need to use a 4-to-1 MUX for the routing unit. Instead, now we can just use 1 single 4-to-1 MUX chip which contains two 4-to-1 MUXs. We needed a lot of 2-to-1 MUXs in our design anyways so having one more usually would not result in one more chip used. Also, if we designed using an 8-to-1 MUX, we have to test all 8 functions and make sure they work, while we only need to test 4 functions if we use the 4-to-1 MUX and make sure the inverting 2-to-1 MUX works as well.

Discuss the design process of your state machine, what are the tradeoffs of a Mealy machine vs a Moore machine?

- The Mealy machine has a lot less states that we have to worry about that Moore machine. However, because it is more compact, we have to be extra careful when considering the inputs and the transition states.

What are the differences between ModelSim and SignalTap? Although both systems generate waveforms, what situations might ModelSim be preferred and where might SignalTap be more appropriate?

- ModelSim is used to generate simulation waveforms that have the inputs already pre-written down in the testbench. Meanwhile SignalTap is used to track waveforms that are happening in real time when we trigger the Execution button on the FPGA board for example. ModelSim is a good way to test and simulate if our project is functional without actually having a physical device. However, we still would want to know if the project is working as we wanted it to on the physical device, so SignalTap can generate actual waveforms that are happening in the FPGA.

Conclusions

In this lab, we learned to build a 4-bit logic processor from a breadboard using a mix of combinational logic, register units, computational unit, routing unit, and control unit. We gained a better understanding of counters, flip-flops, and MUXs after building the circuit. We also learned some basic SystemVerilog and changed a 4-bit processor into an 8-bit one on Quartus and tested it with ModelSim and our FPGA board.